Generics

(http://docs.oracle.com/javase/tutorial/java/generics/index.html)

Generic: same type

What is Generics?

- A technique allows programmers creating general processes on data whose data types are not determined (generic is not used) or they can be determined (generic is used) when they are used.
- A way allows programmer implementing general algorithms which can be used to process multi-type input → Polymorphism.

Objectives

- How we can create a list of arbitrary elements?
- Generics in Java API (java.util pakage)
- Advantages of Generics
- How to create a generic class/ method/ interface
- How is a generic class treated by compiler?
- How to give bounded type parameters?
- Restrictions on Generics

A list of arbitrary elements

- Reference type conformity: fatherRef=sonRef
- The Object class is the ultimate class of all Java class

➔We can create a list of elements which can belong to different classes

➔ **A demonstration:**

```
class Point {
    int x, y;
    Point(int x, int y) { this.x=x; this.y=y;}
    public String toString() {...}
}
public class NonGenericDemo {
    Object[] ar = new Object[100];
    int n=0;
    void add(Object obj){ ar[n++]=obj;}
    void print(){
        for (int i=0; i<n;i++) System.out.println(ar[i]);
    }
    public static void main(String[] args){
        NonGenericDemo obj= new NonGenericDemo();
        obj.add(new String("Hello"));
        obj.add(5);
        obj.add(new Point(9,3));
        obj.print();
    }
}
```

```
run:
Hello
5
[9,3]
```

Generic Classes in java.util

- Almost of interfaces and classes related to lists in the Java API declared as generic.
- Type Parameter Naming Conventions
  - By convention, type parameter names are single, uppercase letters.
  - The most commonly used type parameter names are:
    - E : Element/ K: Key
    - N – Number/ T - Type
    - V - Value
    - S,U,V etc. - 2nd, 3rd, 4th types

- java.lang.Object
  - java.util.AbstractCollection<E>
    - java.util.AbstractList<E>
      - java.util.AbstractSequentialList<E>
        - java.util.LinkedList<E>
      - java.util.ArrayList<E>
      - java.util.Vector<E>
        - java.util.Stack<E>
    - java.util.AbstractQueue<E>
      - java.util.PriorityQueue<E>
    - java.util.AbstractSet<E>
      - java.util.EnumSet<E>
      - java.util.HashSet<E>
        - java.util.LinkedHashSet<E>
      - java.util.TreeSet<E>
  - java.util.AbstractMap<K,V>
    - java.util.EnumMap<K,V>
    - java.util.HashMap<K,V>
      - java.util.LinkedHashMap<K,V>
    - java.util.IdentityHashMap<K,V>
    - java.util.TreeMap<K,V>
    - java.util.WeakHashMap<K,V>

Generics on a List

- Sometimes, we want to create a list with restrictions as elements must belong to some types → Generic
- Generic is a technique which allows a list of arbitrary objects and supports advantages if elements of a list belong to the same data type.

Advantages of Generics

- Generics add stability to your code by making more of your bugs detectable at compile time.
- Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods and limits on parametric types may be declared.
- Code that uses generics has many benefits over non-generic code.
  - Stronger type checks at compile time
  - Elimination of casts.
  - Enabling programmers to implement generic algorithms.

Generics are not used

- The package java.util supports general-purpose implementations which allows lists containing arbitrary elements
- The cost of this flexibility is we may have to use a casting operator when accessing an element.

Generic1.java *

```java
import java.util.Vector;
class Person {
    String name; int age;
    Person(String n, int a)
        { name=n; age=a; }
    void print ()
        { System.out.println( name + ", " + age);}
}
public class Generic1 {
  public static void main(String[] args) {
      Vector v = new Vector();
      v.add (new Person("Hoa", 23));
      v.add (new Person("Tuấn", 27));
      for (int i= v.size()-1; i>=0; i--)
          ((Person)(v.get(i))).print();
  }
}
```

The class **Object** does not have the **print()** method

Output - Chapter08 (run)

```
run:
Tuấn, 27
Hoa, 23
BUILD SUCCESSFUL (total time: 0 seconds)
```

Generics are used

- If all elements of the collection are homogeneous(identical), the generic technique should be used.
- Generics add stability to your code by making more of your bugs detectable at compile time. Casting can not be used.

Generic1.java | Generic2.java

```java
import java.util.Vector;
class Person2 {
    String name; int age;
    Person2(String n, int a)
        { name=n; age=a; }
    void print ()
        { System.out.println( name + ", " + age); }
}
public class Generic2 {
  public static void main(String[] args) {
      Vector<Person2> v = new Vector<Person2> ();
      v.add (new Person2("Hoa", 23));
      v.add (new Person2("Tuấn", 27));
      for (int i= v.size()-1; i>=0; i--)
        v.get(i).print();
  }
}
```

The casting operators are missed.

Output - Chapter08 (run)

```
run:
Tuấn, 27
Hoa, 23
BUILD SUCCESSFUL (total time: 1 second)
```

Using Generics- Syntax

- **Invoking and Instantiating a Generic Type**
  - *Box<Integer> integerBox = new Box<Integer>();*
- **The Diamond**
  - *Box<Integer> integerBox = new Box<>();*
- **Multiple Type Parameters**
  - *Pair<String, Integer> p1 = new Pair<String, Integer>("Even", 8);*
- **Parameterized Types**
  - *OrderedPair<String, **Box<Integer>**> p = new OrderedPair<>("primes", new Box<Integer>(...));*

Using Generics- Syntax

Demo01.java

Source History

```java
package DemoGenneric;

import java.util.ArrayList;
import java.util.Iterator;

public class Demo01 {

    public static void main(String args[]) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        list.add("PHP");
        list.add("C++");
        Iterator<String> itr = list.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
    }

}
```

Output - JavaApplication11 (run)

```
run:
Java
PHP
C++
BUILD SUCCESSFUL (total time: 0 seconds)
```

Using Generics- Syntax

Demo02.java

Source History

```java

package DemoGenneric;

import java.util.HashMap;
import java.util.Iterator;

public class Demo02 {

    public static void main(String[] args) {
        HashMap<Integer, String> list = new HashMap<>();
        list.put(1,"Java");
        list.put(2,"C++");
        list.put(3,"Python");
        list.put(4,"JavaWeb");

        Iterator<Integer> iter=list.keySet().iterator();
        while(iter.hasNext()){
            Integer key=iter.next();
            System.out.println(key +"-" + list.get(key));
        }
    }

}
```

Output - JavaApplication11 (run)

```
run:
1-Java
2-C++
3-Python
4-JavaWeb
BUILD SUCCESSFUL (total time: 0 seconds)
```

Implementing a Generic class

- Syntax:

class name<T1, T2, ..., Tn> {

code

}

```
public class Box<T> {
 // T stands for "Type"
private T t;
public void set(T t) { this.t = t; }
public T get() { return t; }
}
```

Implementing Generic Class

Box.java | UseBox.java

Source | History

```java
package session10;
// Tạo mẫu chung cho Lớp Box
// Trong đó T có kiểu dữ liệu bất kỳ
public class Box<T> {
    private T t;

    public void add(T t){
        this.t=t;
    }

    public T get(){
        return t;
    }
}
```

Box.java | UseBox.java

Source | History

```java
package session10;

public class UseBox {
    public static void main(String[] args) {
        Box<Integer> IntegerBox=new Box<Integer>();
        Box<String> StringBox=new Box<String>();

        IntegerBox.add(10);
        StringBox.add("Hello");

        System.out.println("Integer value: "+ IntegerBox.get());
        System.out.println("String value: "+ StringBox.get());

    }
}
```

Notifications | Output - Java_Basic (run)

```
run:
Integer value: 10
String value: Hello
BUILD SUCCESSFUL (total time: 2 seconds)
```

Implementing a Generic Methods

- *Generic methods* are methods that introduce their own type parameters.
- The type parameter's scope is limited to the method where it is declared.
- The syntax for a generic method includes a type parameter, inside angle brackets, and appears before the method's return type.

```
public static <K, V> boolean equals(Pair<K, V> p1, Pair<K, V>
p2) {
        return p1.getKey().equals(p2.getKey()) &&
                p1.getValue().equals(p2.getValue());
}
```

Implementing Generic Methods

```java
/* Generic class for processing arrays */
import java.util.Arrays;
public class GenericArray <T> {
    public static <T> T get( int i, T[] ar){
        return ar[i];
    }
    public static <T> void output(T[] ar){
        for (T x: ar) System.out.print(x + ", ");
        System.out.println();
    }
    public static <T> void sort(T[] ar){
        Arrays.sort(ar);
    }
}
```

Implementing Generic Methods…

Generic is not used

Generic is used

```
class GenericArrayUse {
    public static void main(String[] args){
        Integer a[]={1,2,3,4,5};
        GenericArray obj1= new GenericArray();
        obj1.output(a);
        System.out.println(GenericArray.get(3,a));
        Double b[]={1.1, 2.2, 3.3, 4.4};
        GenericArray<Double> obj2= new GenericArray<Double>();
        obj2.output(b);
        String list[]= {"you", "love", "I"};
        GenericArray<String> obj3= new GenericArray<String>();
        obj3.output(list);
        obj3.sort(list);
        obj3.output(list);
    }
}
```

```
run:
1, 2, 3, 4, 5,
4
1.1, 2.2, 3.3, 4.4,
you, love, I,
I, love, you,
```

Comparator Interface

- A comparison function, which imposes a total ordering on some collection of objects
- The following demonstration will show you the way to sort a list based on your own criteria: A list of employees will be sorted based on descending salaries then ascending IDs.

Comparator Interface – Demo.

Employee.java | SortSalary.java | UsingEployee02.java

Source | History

```
1
2   package DemoSort02;
3
4
5   public class Employee{
6
7       String ID;
8       String Name;
9       int salary;
10
11      public Employee(String ID, String Name, int salary) {
12          this.ID = ID;
13          this.Name = Name;
14          this.salary = salary;
15      }
16
17      public String getID() {...3 lines }
20
21      public void setID(String ID) {...3 lines }
24
25      public String getName() {...3 lines }
28
29      public void setName(String Name) {...3 lines }
32
33      public int getSalary() {...3 lines }
36
37      public void setSalary(int salary) {...3 lines }
40
42      public String toString(){
42          return ID+"-"+ Name+"-"+ salary;
43      }
44
45  }
```

Comparator Interface – Demo.

Employee.java × UsingEployee02.java ×

Source History

```
1
2   package DemoSort03;
3   import java.util.ArrayList;
4   import java.util.Collections;
5   import java.util.Comparator;
6
7   public class UsingEployee02 {
8       public static void main(String[] args) {
9           ArrayList<Employee> listemp=new ArrayList();
10          Employee e1=new Employee("SE123", "Nguyen Hoai Bao", 5000);
11          Employee e2=new Employee("SE129", "Nguyen Tien Linh", 4000);
12          Employee e3=new Employee("SE120", "Tran Minh Vuong", 3000);
13          Employee e4=new Employee("SE115", "Nguyen Van Toan", 4000);
14
15          listemp.add(e1);
16          listemp.add(e2);
17          listemp.add(e3);
18          listemp.add(e4);
19
```

Comparator Interface – Demo.

```
20          //Su dung lop nac danh Anonymous de trien khai interface Comparator
            Collections.sort(listemp,new Comparator<Employee>(){
22              @Override
                public int compare(Employee t, Employee t1) {
24
25              int d= t.getSalary()-t1.getSalary();
26
27              if(d>0) return -1;// Sap giam dan theo Salary
28              if(d==0) return e1.getID().compareTo(e2.getID());// Sap tang dan theo ID
29              return 1;
30
31              }
32
33          });
34
35          System.out.println("Employee List :");
36          for(int i=0;i<listemp.size();i++)
37              System.out.println(listemp.get(i));
38
39      }
40
41  }
```

Comparator Interface – Demo.

Output - JavaApplication22 (run)

```
run:
Employee List :
SE123-Nguyen Hoai Bao-5000
SE115-Nguyen Van Toan-4000
SE129-Nguyen Tien Linh-4000
SE120-Tran Minh Vuong-3000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Summary

- Generics on methods, classes and collections
- Bounded Type Parameters
- Working with Wildcards
- Working with type erasure
- Generic restrictions